# Multithreading

В простейшем случае процессу соответствует адресное пространство и один управляющий **поток** .

- Причиной использования множества потоков является выполнение большинством приложений существенного числа действий.
- Легкость создания и уничтожения потока (уходит в 100 раз меньше времени).
- Производительность многопоточного приложения.

**Различные потоки** в одном процессе не так независимы,

как **различные процессы**

# Multithreading_threads peculiarity

## Свойства потоков:

- располагают одним и тем же адресным пространством => совместное использование глобальных переменных,
- каждый поток работает с собственным стеком,
- все потоки имеют доступ ко всему адресному пространству своего процесса, даже к пространству стеков друг друга,
- в отличие от процессов, инициированных различными пользователями, потоки работают совместно (в процессе, запущенном одним пользователем).

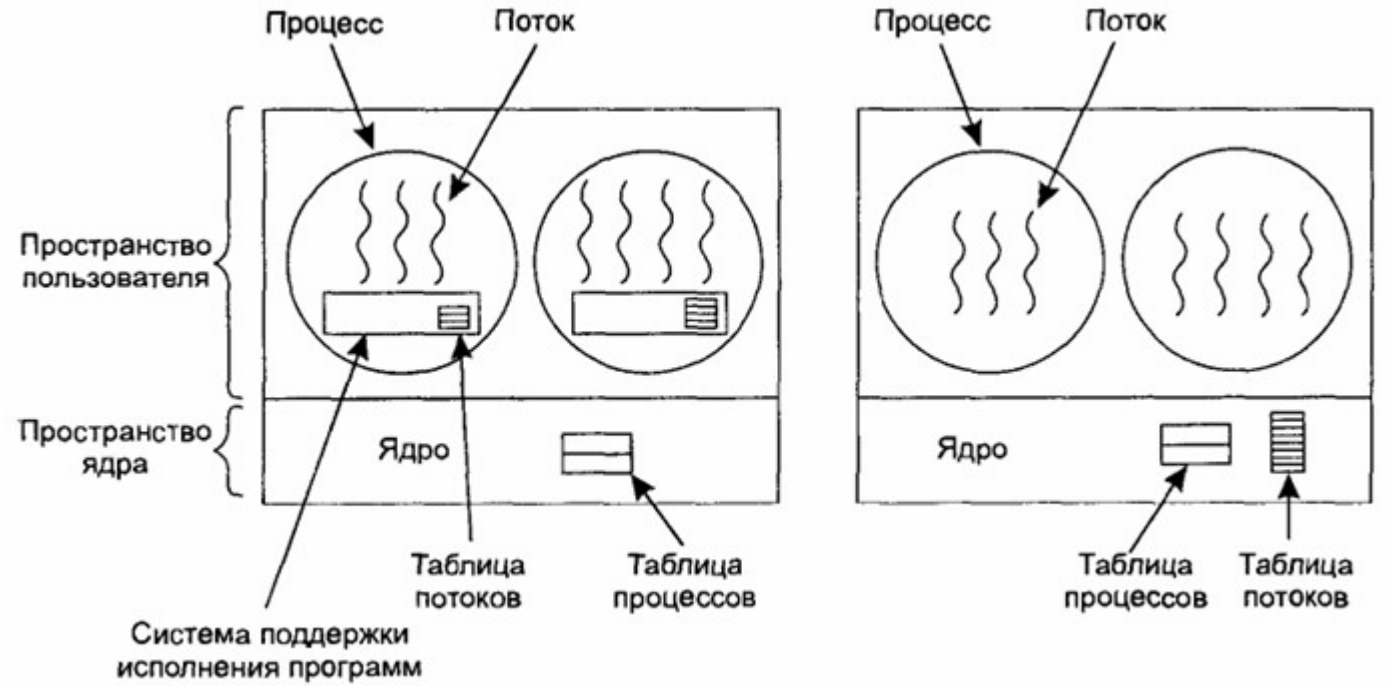
# Multithreading_threads attributes

Потоки имеют

- набор атрибутов в таблице потоков;
- собственный идентификатор;
- набор регистров, включая счетчик команд;
- размер стека, указатель стека;
- параметры планирования и др.

# Multithreading_chart

Способы реализации:  в пространстве пользователя и в ядре



# Pipe_definition

## Конвейеры **pipes**

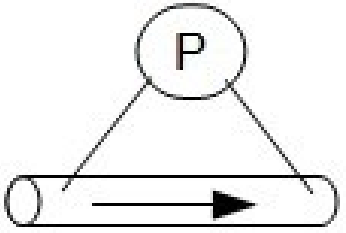
Конвейер – разновидность файла, сохраняющего ограниченный объем данных. Доступ к этим данным организован в манере дисциплины обслуживания FIFO (first-in-first-out).

- Конвейеры в shell запускаются с помощью символа **|** (например, who **|** sort)
- Организация конвейера в приложении системным вызовом **pipe**() .
- Предельный размер буфера конвейера указан в <limits.h> константой PIPE_BUF.

# Pipe_creation related

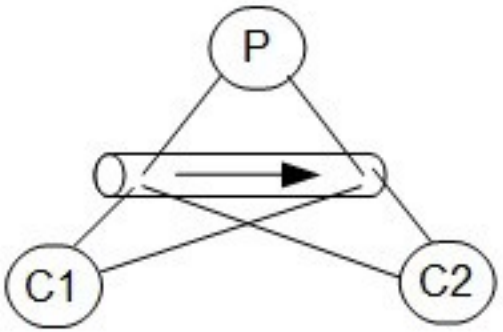
Создание конвейера между процессами-потомками

➔ Родительский процесс **P** вызовом **pipe**() создает конвейер:




➔ Затем порождает два дочерних процесса, каждый из которых наследует оба открытых дескриптора:

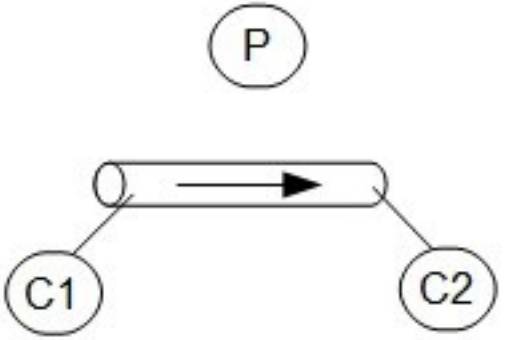
# Pipe_creation related




➔ Потомки **C**1 и **C**2 закрывают дескрипторы, соответствующие тем концам конвейера, в которых они не нуждаются, а на оставшиеся настраивают свой *стандартный вывод* и *стандартный ввод*, соответственно.

# Pipe_creation related

➔ После того, как родительский процесс **P** отключится от канала (закроет оба дескриптора), в системе образуется однонаправленный канал. Данные из *выходного потока* процесса **C**1 будут поступать в *поток ввода* процесса **C**2.



# Pipe_whosortpipe.cpp

```
/* Программа whosortpipe.cpp */
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
main()
{
int    fds[2];
pipe(fds);   /* Создание конвейера */
       /* Один дочерний процесс подключает стандартный ввод stdin к
выходу конвейера
       и закрывает другой конец */
   if (fork()==0){
       dup2(fds[0], 0);  /* перенаправление ввода */
       close(fds[1]);
   execlp("sort", "sort", 0);
   }
/* Другой дочерний процесс подключает стандартный вывод stdout ко
входу конвейера
       и закрывает другой конец  */
   else if (fork()==0){
          dup2(fds[1], 1);      /* перенаправление вывода */
          close(fds[0]);
       execlp("who", "who", 0);
       }
/* Процесс-родитель закрывает оба конца конвейера и ожидает
завершения обоих
       дочерних процессов */
       else{
          close(fds[0]);
          close(fds[1]);
          wait(0);
          wait(0);
          }
}
```

# Pipe_remarks

- Каждая команда (программа), запущенная из командного интерпретатора shell , получает **3 открытых потока**:
  - стандартный ввод (дескриптор **0**)
  - стандартный вывод (дескриптор **1**)
  - стандартный вывод ошибок (дескриптор **2**)

- По умолчанию, все эти потоки ассоциированы с терминалом.

# Pipe_popen() pclose()

Системные вызовы **popen**() и **pclose**() - другой путь создания конвейеров для **исполнения команд** shell.

- Вызов **popen**() генерирует дочерний процесс, который запускает exec()-ом на исполнение команду shell, указанную первым параметром **popen**(). Возвращает значение дескриптора файла (конвейера).
- Второй параметр указывает, как интерпретировать дескриптор файла:
  - “**w**” – родительский процесс может писать данные на стандартный ввод команды shell. Процесс-потомок, выполняющий команду, читает эти данные, как стандартный ввод.
  - “**r**” - родительский процесс читает данные со стандартного вывода команды shell, запущенной дочерним процессом (наоборот).
- Вызовы **pclose**() закрывают конвейер по дескрипторам входа и выхода.

# Pipe_cmdpipe.cpp

```
/* Программа cmdpipe.cpp */
/* Использование команд popen и pclose  */
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<limits.h>
main()
{
FILE  *fin, *fout;
char  buffer[PIPE_BUF};
int    n;
   if(argc<3){
       fprintf(stderr, "Usage %s cmd1 cmd2\n", argv[0]);
       exit(1);
       }
fin  = popen(argv[1], "r");
fout = popen(argv[2], "w");
while((n = read(fileno(fin), buffer, PIPE_BUF))>0)
   write(fileno(fout), buffer, n);
pclose(fin);
pclose(fout);
exit(0);
}
```

# Unnamed pipes weakness

## Недостатки **pipe**

- Невозможность аутентификации процесса на другой стороне (читающий из конвейера  не может знать, кто туда пишет).
- Обмен данными возможен лишь между родственными процессами.
- Не позволяют обмен данными по сети (оба процесса должны выполняться на одном и том же компьютере).

# Named pipes

## Именованные конвейеры **named pipes**

- Используются для взаимодействия **любых** процессов (а не только родственных), наделенных соответств. правами.
- При создании образуют специальные файлы (с маркером **p**).
- Права доступа к конвейеру отображаются в атрибутах файла.
- Могут создаваться как из приложения, так и на уровне shell.
- Конвейеры могут использоваться для организации взаимодействия процессов, работающих в клиент-серверной парадигме.

# Named pipes_shell

## Именованный конвейер на shell

$ mkfifo mynamedpipe

$ ls -l mynamedpipe

prw-rw-r-- 1 xxx xxx 0 сен 14 21:16 mynamedpipe

$ ls -l > mynamedpipe

*Здесь после Enter команда подвисает. Конвейер блокируется до тех пор, пока к его выходу не будет подключена команда (процесс) читающий из него.*

*Далее уже со второго терминала:*

$ cat < mynamedpipe

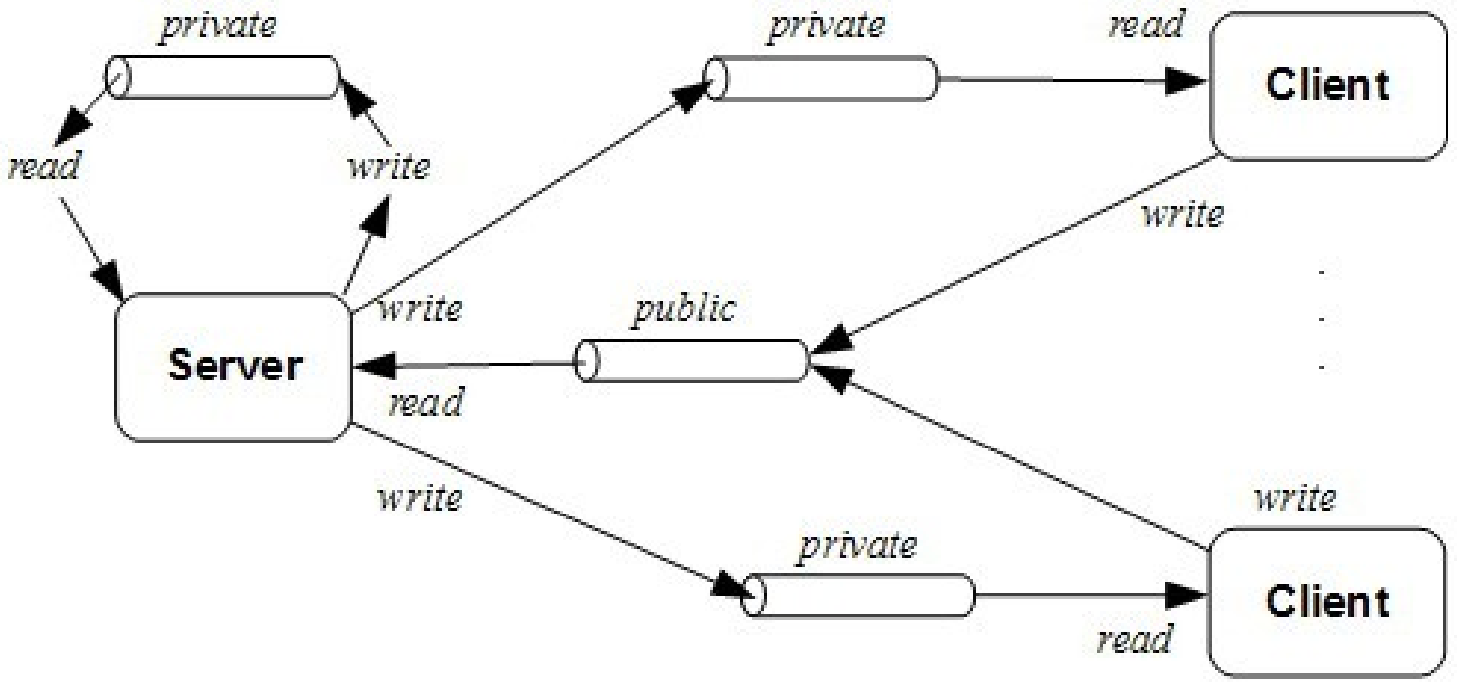
*Здесь после Enter на второй терминал выведется содержимое текущего каталога, а на первом завершится подвисшая команда.*

# Named pipes_application

## Именованный конвейер в приложении

- Системный вызов **mknod**() создает **named pipe** (и не только).
- Первый параметр передает указатель на полное имя.
- Второй задает тип (**S_IFIFO** для очереди FIFO).
- Второй параметр может содержать логическое слагаемое с правами доступа к конвейеру (например, S_FIFO | 0666).
- Третий параметр задает тип устройства (для S_FIFO это 0).

# Named pipes_client server chart



# Named pipes_application

- Процесс сервер запускается в background и создает именованный конвейер PUBLIC, доступный для любого клиента.
- Клиенты запускаются в foreground и каждый из них создает свой конвейер с уникальным именем.
- Клиент в ответ на свой запрос получает с консоли от пользователя команду shell на выполнение.
- Клиент пишет в конвейер PUBLIC имя своего конвейера и команду пользователя.
- Сервер, исполняет полученную команду, используя собственный конвейер, применяет вызов popen().
- Пишет результат команды в конвейер клиента, а тот выводит результат на *стандартный вывод ошибок*.

```
/* Программа pipe_local.h */
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<limits.h>
#include<string.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#define PUBLIC "/tmp/PUBLIC"
#define B_SIZ (PIPE_BUF / 2)
struct message{
char  fifo_name[B_SIZ];
char  cmd_line[B_SIZ];
};
```

# Named pipes_server

```
/* Программа  pipe_server.cpp */
#include<pipe_local.h>
void main(void)
{
int     n, done, dummyfifo, privatefifo, publicfifo;
static char buffer[PIPE_BUF];
FILE *fin;
struct message msg;
       /* Создание очереди типа public FIFO */
mknod(PUBLIC, S_IFIFO | 0666, 0);
        /* Открыть public FIFO на чтение и запись */
if ((publicfifo=open(PUBLIC, O_RDONLY))==-1) ||
    (dummyfifo=open(PUBLIC, O_WRONLY | O_NDELAY))==-1){
   perror(PUBLIC);
   exit(1);
   }
              /* Сообщение можно прочитать из public конвейера */
while(read(publicfifo, (char *) &msg, sizeof(msg))>0){
   n = done = 0; /* Очистка счетчиков | флагов */
   do{                        /* Попытка открытия private FIFO */
       if ((privatefifo=open(msg.fifo_name, O_WRONLY | O_DELAY))==-1
       sleep(3);                    /* Задержка по времени  */
       else{                        /* Открытие успешно */
          fin = popen(msg.cmd_line, "r");/* Исполнение shell cmd,
полученной от клиента */
          write(privatefifo, "\n", 1);    /* Подготовка очередного вывода */
          while((n=read(fileno(fin), buffer, PIPE_BUF))>0){
              write(privatefifo, buffer, n);/* Вывод в private FIFO к клиенту */
              memset(buffer, 0x0, PIPE_BUF);     /* Очистка буфера */
              }
              pclose(fin);
              close(privatefifo);
              done = 1;                  /* Запись произведена успешно */
          }
       }while(++n<5 && !done);

       if(!done)                          /* Указание на неудачный исход */
          write(fileno(stderr), "\nNOTE: SERVER ** NEVER **
                          accessed private FIFO\n", 48);
   }
}
```

# Named pipes_client

```
/* Программа  pipe_client.cpp */
#include<pipe_local.h>
void main(void)
{
int     n, privatefifo, publicfifo;
static char buffer[PIPE_BUF];
struct message msg;
                /* Создание имени для очереди типа private FIFO */
sprintf(msg.fifo_name, "/tmp/fifo %d", getpid());
                /* Создание очереди private FIFO */
if (mknod(msg.fifo_name, S_IFIFO | 0666, 0)<0){
   perror(msg.fifo_name);
   exit(1);
   }
                /* Открытие очереди типа public FIFO на запись */
if ((publicfifo=open(PUBLIC, O_WRONLY))==-1){
   perror(PUBLIC);
   exit(2);
   }
while(1){    /* Зацикливание */
   write(fileno(stdout), "\ncmd>", 6);
   memset(msg.cmd_line, 0x0, B_SIZ);             /* Очистка */
   n = read(fileno(stdin), msg.cmd_line, B_SIZ);        /* Чтение с консоли
shell cmd */
        if(!strncmp("quit", msg.cmd_line, n-1))
        break;                                  /*Завершение? */
   write(publicfifo, (char *) &msg, sizeof(msg)); /* to PUBLIC */
        /* Открытие private FIFO для чтения вывода исполненной команды
(shell cmd) */
        if((privatefifo = open(msg.fifo_name, O_RDONLY))==-1){
           perror(msg.fifo_name);
           exit(3);
           }
        /* Чтение private FIFO и вывод на результата на
стандартный_вывод_ошибок */
   while((n=read(privatefifo, buffer, PIPE_BUF))>0){
        write(fileno(stderr), buffer, n);
        }
        close(privatefifo);
   }
   close(publicfifo);
   unlink(msg.fifo_name);
}
```

# Thanks for your attention

Спасибо за внимание !

vladimir.shmakov.2012@gmail.com